OMB No. 0704-0188

...is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources ...ing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this ...cing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson ...d to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

2. REPORT DATE March, 1991	3. REPORT TYPE AND DATES COVERED ✓ ~
4. TITLE AND SUBTITLE METEORs: High Performance Theorem Provers Using Model Elimination	5. FUNDING NUMBERS DAAL03-88-K-0082 2
6. AUTHOR(S) O. L. Astrachan, D. W. Loveland	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Computer Science Department Duke University Durham, NC 27706	8. PERFORMING ORGANIZATION REPORT NUMBER CS-1991-8
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U. S. Army Research Office P. O. Box 12211 Research Triangle Park, NC 27709-2211	10. SPONSORING/MONITORING AGENCY REPORT NUMBER ARO 25195.9-MA-AI

11. SUPPLEMENTARY NOTES
The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Historically, depth-first (linear) resolution procedures have not fared well in proving deeper theorems relative to breadth-first resolution provers of various types, primarily because of the search redundancy problem. However, we can now demonstrate that the Model Eliminator (ME) procedure, a linear-input resolution-like procedure, may be a superior approach for certain types of problems (generally non-Horn problems). There is a conjunction of reasons why the METEOR provers presently appear so effective. The reasons are: the inherent speed advantage of linear input systems, the sophistication of the WAM architecture in exploiting this advantage, a program written in the language C using tight coding and effective data structures, the speed of the platforms on which they run, and the successful use of different search stragegies. We explore single processor and parallel processor implementations of ME using different versions of interactive depth-first search. Among the theorems we prove are variants of a problem in calculus for which this ME theorem prover is presently the only uniform first-order proof procedure realization that has succeeded in proving these variants.

14. SUBJECT TERMS Linear input proof procedures, Model Elinination first-order proof procedures, parallel implementation, automated theorem proving	15. NUMBER OF PAGES
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102

CS-1991-08

METEORs: High Performance Theorem Provers Using Model Elimination

O. L. Astrachan
D. W. Loveland

Department of Computer Science

Duke University

Durham, North Carolina 27706

METEORs: High Performance Theorem Provers using Model Elimination

O. L. Astrachan and D. W. Loveland*

March 4, 1991

1 Introduction

It has been thought for some time that linear resolution procedures are not the best resolution formats for doing very large unguided searches such as are required for proving significant mathematical theorems. Although very efficient with storage space, the depth-first implementations that take advantage of the linearity seem to preclude use of subsumption and similar devices to reduce redundant search. However, research in logic programming has led to sophisticated architectures giving high-speed performance for Prolog, which is based on a linear input resolution procedure. In particular, the WAM architecture, due to D.H.D. Warren (see [33]) has proven very successful. This has led to renewed interest in Model Elimination (ME) ([18],[19], [20]) which is a linear input proof procedure for full first-order logic, similar to but distinct from resolution. Interest has revived because there are situations where the significantly higher inference rate may offset the removal of some redundant computations.

That this offset can be realized is shown here by applying our theorem prover METEOR successfully to two of Woody Bledsoe's challenge problems [5]. To our knowledge we are (at present) the only other prover besides the STR+VE program of Hines-Bledsoe [6] to find automated proofs for both problems. It is appropriate to note here that theorem provers developed by Woody Bledsoe and colleagues set the standards others try to match; that is partly why we honor him with this volume. Our limited success relative to STR+VE is interesting only because we employ a prover presently devoid of specialized mechanisms for designated predicates. Our failure to prove a third challenge problem shows the limitation of our approach (indeed the limits of pure general first-order proof procedures) but it will allow us to note the potential of selective use of automatically generated lemmas.

*Both authors have received partial support from U.S. Army Research Grant DAAL03-88-K-0082 and NSF Grant CCR-8900383 for the research and writing of this paper

In this paper we report on METEOR, a parallel implementation of ME (or-parallelism, to be precise) and compare it to the sequential version of the same inference engine as well as comparing the implementations with others reported in the literature. (METEOR is derived from Model Elimination Theorem proving with Efficient OR-parallelism.) The significance of parallel ME in future theorem provers stems from the availability (and even low cost) of parallel machines and the decomposability of theorem proving tasks, including search itself. The rapid progress of chip technology assures us of low-cost access to powerful parallel computers in the near future. The possibility of many specialized (and competing) components in a theorem prover suggests that parallel computers may be the preferred vehicles in the future. (Clearly, distributed processing is also appropriate here.) The immediate focus is to demonstrate that the inference rate is appropriately enhanced by parallelization and that added gain is possible by utilizing the flexibility of computation that parallel computation suggests.

For many years interest in ME was kept alive by Mark Stickel, who has demonstrated that the WAM architecture can be adopted to ME and yield a very high inference rate in this setting as well. This relationship to Prolog architecture is even recognized by the title in the recent theorem prover(s) developed by Stickel: the Prolog Technology Theorem Prover ([29], [30], [32].) When we began our project, Stickel's work was the sole guide for comparison. However, as we write this summary of an implementation of parallel ME, several years after initiation of the project, we see that we were not alone in seeing new opportunity in exploiting Model Elimination. Two other projects (that we know of) simultaneously have developed parallel ME implementations using the concepts employed in Prolog implementations, indeed using ideas from implementations of parallel versions of Prolog. PARTHENON, developed at CMU by Bose, Clarke, Long and Michaylov ([7] [8]) employs a modification of the SRI architecture for or-parallel Prolog designed by D.H.D. Warren [34]. PARTHENON has been run on a Encore MULTIMAX computer, using up to 16 processors, and on the IBM RP3, using up to 60 processors. Both systems are MIMD shared-memory systems. PARTHEO, developed at TUM (Munich) by Schumann and Letz [28], consists of a uniform network of sequential processors for ME communicating by message passing. The individual ME processors use an extension of the WAM architecture.

Our implementation has much in common with both of the abovementioned implementations for reasons given earlier; all have utilized the WAM architecture and the very close design relationship between Prolog (SLD-resolution) and ME. Our implementation is closer to PARTHENON in that it was originally based on Warren's SRI model and also runs on an MIMD shared-memory machine. (The current implementation is closer to the or-parallel system developed at SICS [1] than to the SRI model.) We use a BBN Butterfly GP1000 with up to 30 processors. We have also adapted the prover to run on a network of Sun SPARC stations each of which is a significantly more powerful inference engine than one node of the Butterfly.

Perhaps the most significant design difference with both other systems is our task allocator or scheduler. Whereas the other systems use a local task "stealing" mechanism, we use a top-level public sector/private sector task dispensing system (as in [21]) we call a SASS-pool, for Simultaneous Access of Selectively Shared objects pool, a type of concurrent pool. The notion of concurrent pool has been extensively studied by Ellis and Kotz (see [15]). This paper will focus on the differences between this system and the other implementations.

The most significant discovery from our experimentation with the METEOR project is the effect of different depth measures on the proof search size. The proof search uses iterative deepening, the now standard method of realizing completeness in a depth-first search procedure [14]. By iterative deepening we mean expanding the search tree fully to a specified depth, incrementing the depth bound and reexpanding the search tree, and continuing this until a proof is found or the search is terminated. Within this basic search strategy are significant variants determined by the measure of depth used. Using the number of inferences in a deduction as a depth measure, as done by PTTP [32] and PARTHENON [8], one is assured of a shortest proof if any proof is found. However, we have discovered that the original depth measure suggested for ME [18] very often finds a proof in a dramatically shorter time, although not necessarily a shortest proof. The *SETHEO/PARTHEO [28]* provers seem to use this measure uniformly, but with no reported data on comparison of depth measures. We have also discovered two different combination depth measures of interest, one that is a compromise of the two depth measures but is uniformly better than using the number of inferences as a depth measure, and one that is often strikingly good for reasons we do not fully understand.

After a presentation of the architecture of METEOR, we present performance figures on a collection of benchmark problems run on several other theorem provers, comparing previous performances with data from our sequential prover and from the METEOR parallel prover. We also provide performance figures on three of five Bledsoe challenge problems regarding the continuity of the sum of two continuous functions [5]. We begin with a brief review of the ME procedure, the WAM architecture and the METEOR architecture, a parallel version of the WAM architecture.

2 Background

We first outline the Model Elimination (ME) procedure. We assume the reader is familiar with basic resolution terminology. For such terminology, see Chang & Lee [10] or Loveland [20]. The latter book gives a thorough introduction to ME.

The ME procedure utilizes many of the same mechanisms as resolution, and can be roughly regarded as a linear input resolution procedure with the addition of an ancestor rule we call (ME) reduction. (However, the resultant resolution procedure corresponding to ME is not a linear input procedure. Linear input resolution procedures are complete only for the renamable Horn sets at the ground level.) The interest in the procedures is due to retention of the linear input property while providing a complete proof procedure. By a *linear* (resp., *input*) procedure we mean a procedure all of whose deductions are linear (resp., input) deductions. A *linear deduction* is a deduction where each line is derived from the preceding line (often using auxiliary information). An *input deduction* is a deduction where each line uses as auxiliary information only a given (i.e., input) formula. For example, an input resolution deduction restricts the resolution inference rule to one derived clause (at most) and one given clause as parents to the resolvent. It follows that input deductions are linear, since two lines of deduction can never join, but we often use the phrase *linear input* to emphasize the linearity (and because the phrase is in common use).

ME has two basic inference rules, the extension rule and the reduction rule. The rules manipulate formulas called chains, defined below. The extension rule (also defined below) is very similar to the resolution rule, using as parent formulas a derived chain and a given chain, and producing a chain. Reduction takes one chain as input and produces a modified chain. Hence the system has a (linear) input form.

ME is a refutation procedure, taking as pre-input a clause set prepared as is done for resolution. The actual input is a set of chains of a certain type. A *chain* is a sequence of literals; each literal is one of two types, A-literal (for ancestor) or B-literal. The ordering that determines acceptable sequencing is determined by the user subject to constraints given below. An input chain is a sequence of B-literals composed of the literals from a clause of the given clause set. The ordering of each input chain is determined entirely by the user except that for each literal in a given clause there must be a chain with that literal leftmost. Thus, for an n-literal given clause there are n input chains. Chains are accessed only by the leftmost literal so this restriction states that every literal of every given clause is accessible. (Whether or not n copies of an n-literal clause actually exist is an implementation matter.) (In [20], and in the earlier papers introducing ME [18], [19], chains had the rightmost literal as the accessible literal. That ordering is more convenient for presenting deductions. However, Prolog conventions utilize left access, and, as ME is now implemented using augmented Prolog architectures, we accommodate these conventions.)

Definition 2.1 *A chain is* admissible *if and only if (iff):*

1. *complementary B-literals are separated by an A-literal;*
2. *no B-literal is to the left of an identical A-literal;*

3. no A-literal is identical or complementary to another A-literal; and

4. the leftmost literal is a B-literal.

We need only deal with admissible chains. Any derived chain not admissible can terminate that search branch. These conditions, particularly condition 2, make useful pruning rules.

We give the two inference rules. Recall that two literals are complementary iff one is the negation of the other. By mgu we mean the "most-general unifier".

Definition 2.2 Extension*: Given admissible chain C_1 and input chain C_2 such that the leftmost B-literal l_1 of C_1 and the leftmost B-literal l_2 of $C2$ can be made complementary by mgu σ, the resultant chain is $C_1\sigma$, with $l_1\sigma$ promoted to an A-literal augmented on the left by $C_2\sigma-\{l_2\sigma\}$, with $C_2\sigma-\{l_2\sigma\}$ reordered as the user wishes. Any leftmost A-literals are removed.*

Definition 2.3 Reduction*: Given admissible chain C with leftmost B-literal l_1 and A-literal l_2 (anywhere in C) such that l_1 and l_2 can be made complementary by mgu σ, then the resultant chain is $C\sigma - \{l_1\sigma\}$ with any A-literals leftmost in $C\sigma - \{l_1\sigma\}$ removed.*

Example: If $C_1 = Pxa[\neg Qy][Pf(x)b]Qx$ and $C_2 = \neg Paa$, then the extension of C_1 using C_2 is Qa, with intermediate step $[Paa][\neg Qy][Pf(a)b]Qa$ before the A-literals (in brackets) are removed as being leftmost (i.e. to the left of all B-literals).

Example: If $C = Pax[\neg Qb][\neg Pyz]$ then reduction can apply, yielding the empty chain, from intermediate step $[\neg Qb][\neg Pax]$ whereupon the A-literals are removed as being leftmost in the intermediate chain.

An ME refutation of a given clause set is a sequence of steps using the inference rules of extension and reduction from the set of associated input chains and with the last entry the empty chain, denoted $\Box$.

We note that there is no equivalent to factoring, although that can be added. SL-resolution (Kowalski & Kuehner [16]) is essentially ME with factoring. Incidentally, the selected literal capability of SL-resolution is present in the ME procedure under a different guise. In ME the user orders the input clause (freely) in the extension derived chain according to the order of selection desired. Literals between A-literals can be reordered at any point, with the justification that the resulting order could have been selected initially, if desired.

Before giving a short ME refutation we mention the lemma capability of ME which has never been extensively explored. (Lemmas were used in [12] in an unconstrained manner, and found generally not helpful without further control.) There is a device for creating derived clauses during the search process that can be used to shorten proofs. Since these lemmas are not needed for completeness (unlike derived resolution resolvents) very strong restrictions can be applied regarding retention. The lemmas may by used to form new input chains. (We feel that most often only one-literal lemmas would be retained.) The use of lemmas and related devices ("caching") have the potential to greatly reduce the search needed to achieve refutations if the right mechanisms for use can be found. We are exploring this now.

A unit lemma may be created any time an A-literal is removed from the left of an intermediate chain during extension or reduction, provided that no reduction previously has used an A-literal to the right of the discarded A-literal. The unit lemma is simply the complement of the removed A-literal, as currently instantiated. (This is a simplification of the full rule for lemmas, given in [19] [20].)

In Figure 1 we present an ME refutation given in [20]. No ordering rule is needed here since the longest chain has two literals. The input chains are never explicitly listed, only the given clauses are listed; the input chains are irrelevant except conceptually because it is only the ordering after extension that really matters.

This ME refutation illustrates lemmas, and also that a chain with complementary B-literals separated by A-literals cannot be discarded. (Cf. the definition of admissible chain given earlier.)

(The label 1b means the second literal of clause 1.)

1.	$Px\neg Qy$	given clause
2.	$\neg PxRy$	given clause
3.	$\neg Pa\neg Ry$	given clause
4.	$QxPa$	given clause
5.	$Py[Qx]Pa$	extension with 1b
6.	$Rx[Py][Qx]Pa$	extension with 2a
7.	$\neg Pa[Rx][Py][Qx]Pa$	extension with 3b
8.	Pa	reduction

Unit lemmas formed:
$\neg Pa$
$\neg Qx$

(Note: If lemma $\neg Pa$ is used for extension then $\square$ is immediately derived)

9.	$\neg Ry[Pa]$	extension with 3a
10.	$\neg Px[\neg Ry][Pa]$	extension with 2b
11.	$\square$	reduction

Figure 1: An ME refutation

3 The Warren Abstract Machine

The success of Prolog as a logic programming language is due both to the declarative/procedural nature of logic as a programming language and to the efficiency of current implementations of Prolog. This efficiency is due to the input nature of Prolog deductions and to current realizations of such input procedures. This latter efficiency is due in large part to David H. D. Warren's design of an abstract machine for executing Prolog programs [33]. Most commercial and research Prolog implementations are based to a large degree on the WAM (Warren Abstract Machine) as are the Prolog technology class of theorem provers ([32],[8],[2]).

The WAM consists of both an architecture and an instruction set. Many Prolog implementations compile Prolog programs into WAM instructions (or some variant thereof) which can be run on different machines (e.g., Sun, VAX) by using different back ends for the Prolog compiler/interpreter. The architecture of Warren's virtual machine is designed to take advantage of traditional von Neumann architectures while optimizing execution of Prolog.

The architecture of our prover is based on that of the WAM although we do not compile a set of clauses into some variant of the WAM instruction set. Instead, we use the concepts underlying the design of the instruction set to compile the clauses into a data structure that is tuned to our implementation of parallel Model Elimination.

3.1 The WAM architecture

We give a brief outline of the WAM architecture. For a more thorough explanation see [33],[22]. The WAM is a stack based machine. In addition to the code area (in which compiled Prolog procedures are stored) there are three stacks: the term stack, the trail stack and the runtime stack.

The term stack is often referred to as the heap. Although the first Prolog implementations employed structure sharing [9] for storing bindings of terms, current implementations use copy on use. When a term such as `f(X,g(a,Y))` is substituted for a variable during unification, a copy of the term is made on the term stack. If subsequent deductions cause this binding to be incorrect (and backtracking occurs), the space in the term stack will eventually be reclaimed. Note that the skeleton of a term as it appears in an input clause may be stored several times (with, perhaps, different instantiations of the variables in the term) on the term stack.

The trail stack is used to trail bindings of variables that may need to be undone when backtracking occurs. In a sequential setting, only the addresses of variables that are bound are stored in the trail stack. If backtracking occurs, these addresses are used to restore the state of the bound variable to unbound. In the SRI or-parallel extension to the WAM, both the address of a variable and the value bound to the variable are trailed. This is necessary since different processors may bind different values to the the same variable. Unlike the sequential WAM in which variables are associated with an address, in the SRI extension to the WAM variables are associated with an index in a binding array (and an address). The use of the binding array enables different processors to bind values to the "same" variable by storing the value in the processor's binding array. In our original prover we also used such a binding array. Our current prover, however, uses the model of the sequential WAM in a parallel setting and no binding array is used.

The runtime stack stores both environments and choice points. An environment consists of the variables associated with a given clause and the bindings of those variables. When clauses are compiled, a choice point stores the values of the WAM registers that need to be restored when backtracking occurs. In our setting, a choicepoint stores these values as well as storing the information necessary to make the next "choice" i.e., which chain to try next. We thus fold the code area into the runtime stack, specifically into the choice points that comprise the METEOR runtime stack.

3.2 METEOR Architecture

The data structure diagrammed in Figure 2 shows how a clause set is compiled into a structure used by METEOR. Each chain corresponding to a clause in the clause set is represented in the section labeled *Chain Table*. Recall that in ME each literal of a clause is a candidate for the extension operation. Consequently, a given n-literal clause is represented n times in the chain table: once with each literal as the head (left-most literal) of a chain which corresponds to the clause. Each such instance of a clause is preprocessed to distinguish the first occurrence of each variable (for efficient operation of the occurs check) and indexed according to the first argument of the head literal.

Pointers representing each chain with a given literal as head reference another pointer (the *chain list pointer* section) according to the index assigned to the the head literal's first argument. This level of indirection is essential as it avoids duplicate storage of a chain (each chain must "appear" in the *all* section of the chain table to match a goal whose first argument is a variable as well is in the hash bucket corresponding the chain's first literal) while permitting alternative chains with the same head literal to be accessed sequentially.

The section labeled *.hain list* stores information about a chain that is used during an extension operation. The number of variables in the chain is used in allocating an environment for the chain. The number of "body literals" (the term is derived from Prolog terminology) represents the number of new subgoals that must be solved if a chain is used for extension. This can be used to prune the search tree early using certain depth measures (see below). The literals that comprise a chain are accessed sequentially by following the head pointer as shown in the diagram.

3.3 METEOR Choice Points

At any time during a run of METEOR, the runtime stack stores information that allows the current deduction to be reconstructed. The stack is comprised of choice points, one choice point for each node of the current search tree. If the active branch of the search tree fails for some reason (e.g., depth bound is reached or no matching clauses are found), METEOR backtracks and tries an alternative extension or reduction. When run in parallel, the stack of each processor is divided into a public and private section in the same manner as [21]. This division is realized by putting one choice point from each processor's runtime stack into a publically accessible data structure we call a SASS-pool (Simultaneous Access of Selectively Shared objects.)

Access to this pool is supported by *CREW* (concurrent read exclusive write) locks [3] which permit simultaneous access to choice points while ensuring atomic conversion of a choice point from private to public status. When a "reading" or "stealing" process gains access to a public choice point, work is transferred to the reading process without interrupting the process from which the work is taken. By utilizing the atomic operators available on the Butterfly we have been able to implement these locks so that, on the average, less than one one-thousandth of the total running time is spent acquiring locks for problems with sufficient parallelism. For full details on the locking mechanisms see [2].

3.4 Transferring Work

How work is transferred, from what part of the search tree the work is transferred, and the granularity of the work transferred are important details in an or-parallel system. In METEOR, we transfer work from nodes near the root of the search tree. In the original version of METEOR based on the SRI model, transferral of work included copying part of the global stack from the processor from which work is taken. In addition, a processor's logical stack physically resided on several processors. Although this model supports parallelism to some degree (it is used in PARTHENON and some models of or-parallel Prolog), we feel that current trends in parallel architectures will emphasize NUMA (non-uniform memory access) architectures and that such architectures preclude this approach.

The Butterfly GP1000 machine (on which METEOR has been developed) is a NUMA machine on which local memory accesses are roughly ten to fifteen times faster than remote accesses. Current trends towards cache based machines will make this penalty even more severe on other architectures. Developments in distributed computing also preclude an approach based on physically shared stacks. With these trends in mind, we have abandoned the SRI model and implemented an architecture in which stacks are not physically shared. Instead, when processor P steals work from processor Q, the stack of processor Q is reconstructed locally on processor P. In METEOR, the stack is not copied, but is recreated by copying the sequence of extensions and reductions that led to the state from which work was taken from processor Q. We have adopted this approach, rather than an approach of physically copying stack information, because recreating a stack is an extremely fast operation given our architecture. In addition, this approach has enabled us to implement a distributed version of METEOR on a network of Sun SPARC stations. The distributed version runs in two modes: a shared memory approach using the common file system as a shared pool, and a message passing approach using UNIX sockets. The MUSE or-parallel Prolog system [1] also uses a non-shared stack approach, but physically copies the stack and attempts to optimize transferral of work so that stack copying is minimized. PARTHEO [28] uses a system like ours in which stacks are recreated locally. Kumar [17] discusses the parameters of a system that affect whether copying should be preferred to re-creation.

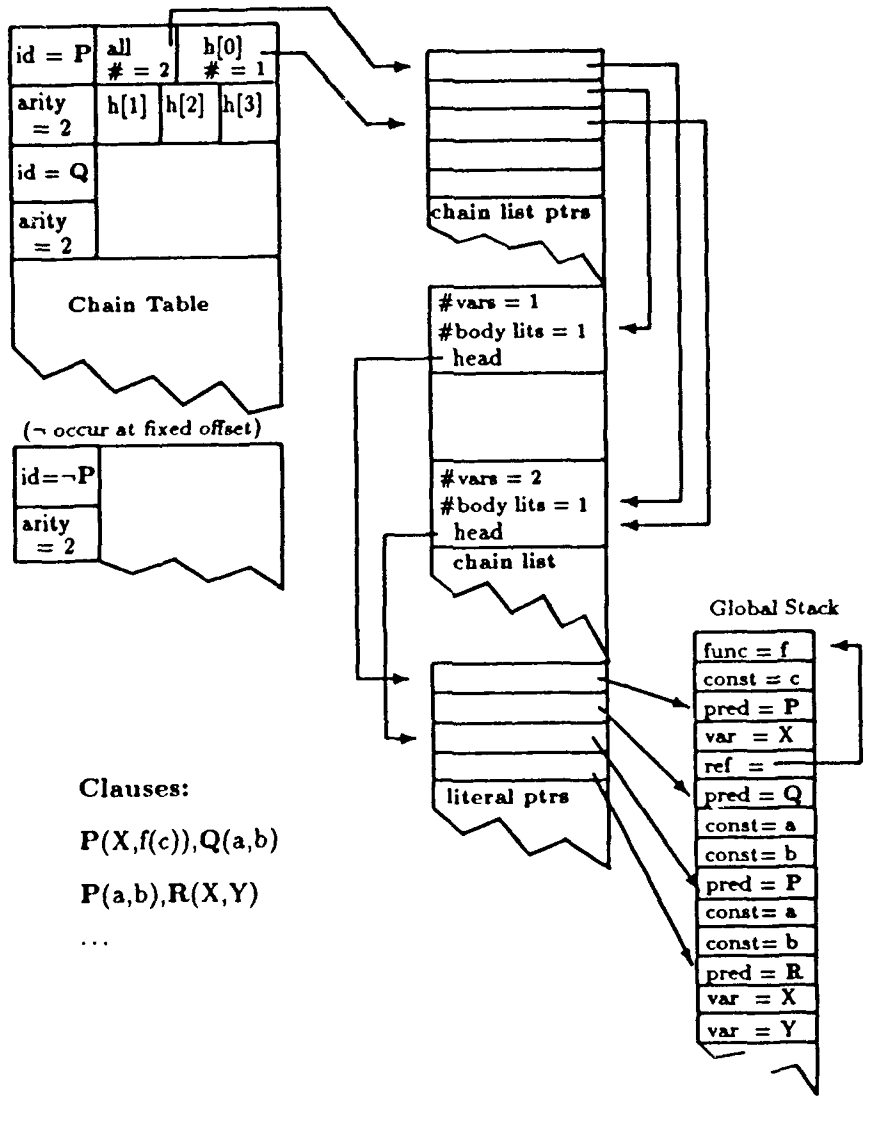

Figure 2: Principal data structures in METEOR

4 Discovering a Proof

In proof procedures based on Model Elimination, the search tree can be viewed as an AND-OR tree [24]. Each extension operation with an n-literal input chain yields an AND node in which there are $n-1$ branches (one for each literal l other than the head literal.) Each of these branches links to an OR node from which there is a branch for each input chain whose head literal matches l (that may lead to an extension operation) and a branch for each matching ancestor literal (that may lead to a reduction operation.) Thus alternating levels of the search tree consist solely of either AND or OR nodes with the OR nodes corresponding (loosely) to a WAM choice point. METEOR and other or-parallel provers attempt to explore the OR nodes of this tree in parallel. A successful search results in a proof tree which consists of AND nodes, several such trees are diagrammed below (3,4,5.)

To ensure completeness, most theorem provers using depth-first search use iterative deepening as a search control strategy. METEOR (as well as PTTP and PARTHENON) employs an iterative form of the A* algorithm called IDA*.

In each iteration of the A* algorithm, nodes of lowest cost are chosen for expansion. Expansion of a node results in the children of the expanded node becoming candidates for expansion in the next iteration of the algorithm and requires the calculation of the cost of each child. The cost of a node, $f(n)$, is determined by $f(n) = g(n) + h(n)$ where g represents the cost of reaching node n and h represents the estimated cost of reaching a goal node from n (the empty chain in an ME deduction.) Like other best-first searches, A* requires exponential storage in practice [25] which makes it prohibitive. IDA* combines the linear storage of depth-first search with the optimality of A*. At each iteration of IDA*, nodes are expanded according to the cost function $f(n)$ as in A*, but the nodes are expanded in a depth first manner (with backtracking) until a threshold is exceeded. If no goal nodes are expanded (the search is not successful), the threshold is incremented (to the minimum of all costs that exceeded the previous threshold) and the search continues.

(Strictly speaking an IDA* algorithm is not used since node selection is controlled by the order in which clauses appear in the input file.)

In the class of theorem provers examined here, both g and h are computed from the same measure: the depth of a deduction. Different depth measures, however, can yield drastically different search times for a given problem.

4.1 Depth Measures

The depth measure employed in PTTP and PARTHENON is based on the number of inferences (extensions and reductions) used to reach a node of the search tree. For any node n we have

$$g(n) \;=\; \text{total number of extensions and reductions to reach } n$$

$$h(n) = \text{number of B literals in the chain at node } n$$

To see that $h(n)$ is an admissible measure (i.e., never overestimates the cost to a goal node) note that the conversion of each body literal introduced by an extension operation to an A-literal requires at least one inference (either a reduction or extension with a unit chain.) This measure will ensure that if a proof is found, the number of inference steps in the proof is minimal. We call this depth measure *cumulative inference* depth, or *cumulative* depth. Note that both g and h (indirectly) are defined in terms of the total number of inferences in a proof.

In many contexts we are concerned with finding proofs of minimal length. At times, however, we may be concerned with minimizing the time to find a proof and may be willing to sacrifice minimization of proof length for a "quicker" proof search. This leads us to also consider a different depth measure, one originally proposed in [18], which attempts to minimize the depth of the search tree. In this case for any node n we have

$$\begin{aligned} g(n) &= \text{number of A-literals in the chain at } n \\ h(n) &= 0 \end{aligned}$$

Note that the number of A-literals in the chain at n is the length of the path from goal to n in the (potential) AND proof tree. We assign h the value 0 since it is possible for each literal introduced at node n to be "solved" by the reduction operation which does not introduce a new A-literal. This depth measure ensures that the maximum number of A-literals in any chain of the deduction is minimized which corresponds to minimizing the depth of the AND-OR search tree. We call this depth measure *ME-depth* since it is the measure outlined in the original presentation of Model Elimination [18].

To see how these different measures can affect the search for a proof, consider the proof trees of two different group theory problems (wos10 and wos1) shown in Figures 3 and 4. The clause sets for these problems are given in section 6 (in the search tree each number refers to the clause number in the input file.) The ME-depth proof tree for wos10 indicates a proof is found at ME-depth four that consists of 16 inferences. There is, however, a ten inference proof found as shown in the cumulative inference depth proof tree. Comparison of the times taken to find these respective proofs (see Table 9) shows that the ME-depth proof takes 12 times as long to find.

In contrast, the ME-depth proof tree for wos1 indicates a proof is found at ME-depth three that consists of 10 inferences. The cumulative inference depth proof tree also consists of a 10 inference proof, but this proof takes roughly 8.5 times as long to find as the ME-proof.

In terms of time taken to find a proof, these problems are relatively simple (there is a large class of problems whose proofs are found in under 1 or 2 seconds by all provers of the PTTP class, we do not consider these here.)

Even so, we gain or lose almost a factor of 10 depending on our choice of depth measure for these problems. For "harder" problems (problems for which it takes more time to find a proof) this difference can be even more pronounced — so much so that the choice of depth measure can determine whether a proof is found.

Given the differences in time to find a proof using the different depth measures, the question naturally arises as to whether a choice of depth measure can be made *a priori*. As we shall see, there are *a priori* reasons for choosing one depth measure in lieu of another based on a static evaluation of the input clause set. We have also developed a heuristic depth measure that combines both ME-depth and cumulative inference depth. We consider the combined measure first.

The obvious methods of combining the two methods in an iterative deepening search involve searching to a given level of the search tree (ME-depth) while constraining the number of inferences permitted in the potential proof tree. As a heuristic, we have chosen to constrain the number of A-literals to be half the number of inferences allowed in exploring the search tree. We call this depth measure ***heuristic depth***. As seen in Tables 9 and 12, this heuristic depth yields uniformly faster search times than cumulative inference depth on the examples run. For several problems, however, using ME-depth yields drastically shorter search times. We are currently investigating using a dynamic combination in which the number of A-literals is a function of both the branching factor in the search tree and the number of inferences.

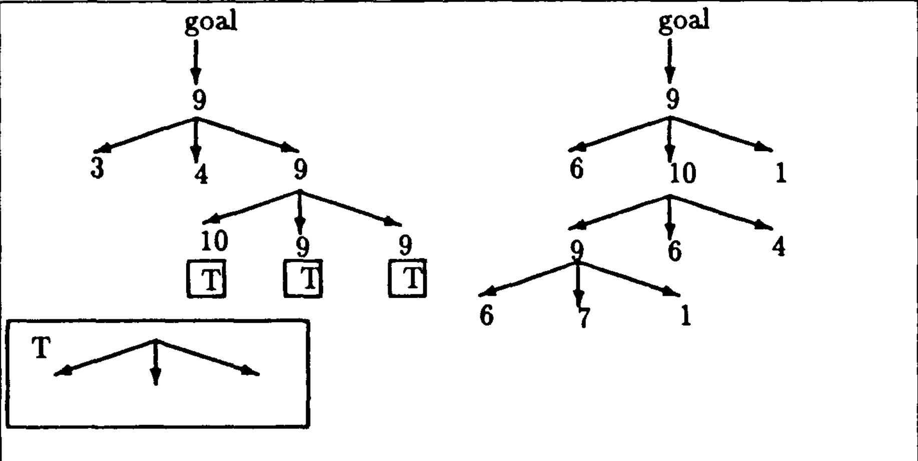

Figure 3: Proof trees for wos10 (ME-depth on left, cumulative depth on right). The box labeled T represents three unit chain extensions.

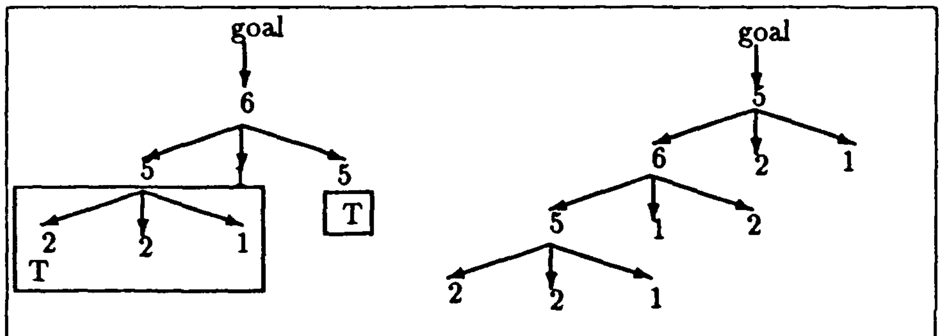

Figure 4: Proof trees for wos1 (ME-depth on left, cumulative depth on right). The box labeled T represents the same extensions with three unit chains.

4.2 An *a priori* choice of depth bound

There is a class of problems for which a static evaluation of the input clause set can suggest that ME-depth is the appropriate depth measure. Consider a clause set which contains an n-literal clause for some "large" n (e.g., $n \geq 4$) or several such clauses. If such clauses are used several times in the proof tree, the tree will tend to be "bushy" i.e., several nodes will have a large branching factor. This is often an indication that the use of ME-depth will yield faster search times. Consider, for example, Schubert's Steamroller problem as formulated in [31]. The version of the problem suggested as the standard has a five literal goal and an eight literal clause among the input clauses. Although several forward chaining provers succeed quickly with this problem, the PTTP class of provers does not succeed when using cumulative search depth. Using ME-depth, however, (or rollback reduction — see below) METEOR finds a proof very quickly as indicated in Table 13. Examination of the proof tree shown in Figure 5 indicates that a proof tree of depth four exists, but that this tree consists of 69 inferences (all extensions). (Each □ in the figure represents an input chain that matches with either 1 or 2 unit clauses as indicated.)

As another class of problems for which ME-depth is well suited we consider the pigeonhole family of problems P_n, where P_n is a clausal form of the statement that $n+1$ pigeons cannot fit in n holes. This class of propositional problems is "hard" in the sense that the number of clauses in P_n is $O(n^3)$, but the shortest resolution refutation of P_n consists of $\Omega(c^n)$ clauses for some constant c ([13]) i.e., the size of the proof tree is exponential in n. The proofs of these theorems depend heavily on the reduction operation. For P_5 we note that the proof tree has depth 13 and that there are 4,178 inferences of which 1,568 are reductions (the proof is found in 25.67 seconds, see Table 13.)

In comparing METEOR runtimes on the pigeonhole problems with results given for other provers (see Table 8) we have found a discrepancy in the use of the parameter n in the pigeonhole problem P_n. Some references ([23], [27])

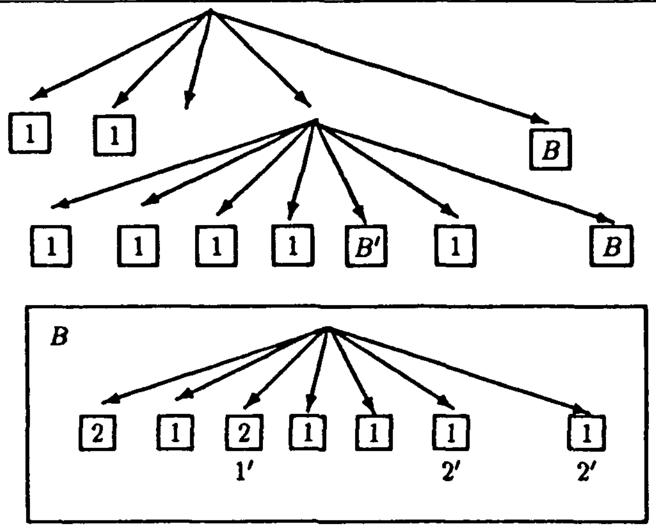

Figure 5: Proof tree for Shubert's Steamroller

use P_4 to refer to the same problem referred to by others as P_3([4].) We have adopted the meaning given in [13] and [4] which is the meaning given in the original presentation of the pigeonhole problem ([11]).

4.3 Rollback Reduction

As part of our examination of different depth measures we have discovered (somewhat serendipitously) a new depth measure which applies only to non-horn clause sets. In a sense this measure combines cumulative inference depth and ME-depth. With this measure, a depth is associated with each literal in a chain as well as with the chain itself. The depth of the initial chain (and the literals that comprise this chain) is zero. The depth of a successor chain depends on whether the chain is formed by extension or reduction. If the depth of a chain C is d, and the successor chain C' is formed by extension, then the depth of C' is $d + 1$ and the depth of each literal (if any) introduced by the extension is also $d + 1$. Thus for extensions, depth is calculated as it is using cumulative depth. If C' is formed from C by reduction, however, the depth of C' is the depth of the accessible (left most) literal of C'.

Since the depth of a successor chain can decrease after a reduction operator (and never increases) we call this depth measure *rollback* reduction.

We have tried this measure on several non-horn problems and it is uniformly better than using cumulative inference depth on these problems. Using rollback reduction does not always result in a shortest proof. However, it appears that by allowing more reductions than would be allowed using cumulative inference depth that deeper proofs are found more quickly. For example, wos4 (another group theory problem) has a minimal length proof of 13 steps found in 1,083 seconds using cumulative depth (see Table 13) and an 18 step proof found in 0.78 seconds using rollback reduction.

The results for several non-horn problems are given in Table 13. For those problems for which no cumulative depth statistics are given, METEOR failed to find a proof using cumulative depth.

4.4 Weighted depth

The depth measures we have mentioned may be set by runtime parameters when METEOR is invoked. We have also implemented a depth measure that may be used in conjunction with either cumulative depth or ME-depth, but which requires that input clauses be annotated with a weight. This weight should reflect the user's concept of how often a chain corresponding to the clause is used on any AND branch of the search tree ("heavy" clauses will be used less often.) For example, in a group theory problem (e.g., wos1 and wos10, see pages 25 and 26) the axioms of associativity have many literals, none with ground terms. Unconstrained use of such clauses can result in chains that are not part of any proof i.e., in searching dead ends in the search tree. One would thus weight these clauses more heavily than ground unit clauses.

When clauses are weighted and the appropriate runtime parameters are set to invoke weighting, the weight of a node N is the sum of the weights of the clauses used in the proof on the path from the root of the search tree to N. Typically, a weight bound is set by the user and this bound is used in conjunction with clause weighting to prune branches of the search tree that exceed this bound. If no proof is found with a given weight bound, the bound is incremented (automatically) and the search continued. We call this depth measure *weighted ME-depth* or *weighted depth*.

We use weighted depth in conjunction with either cumulative depth or ME-depth, not as a substitute for either. We have assigned weights to the clauses for many of the problems for which statistics are given in the tables below. For a given class of problems (e.g., wos problems, bledsoe problems), the weights assigned to common clauses are the same across problems in the same class. Statistics for the use of weighted depth with several problems are given in Table 9. The choice of weight bound and weight increment (the amount by which the bound is incremented on each search iteration) can have a profound effect on the search time. We choose the bound based on the weights of the heaviest clauses and an increment that permits progressively more instances of these heavy clauses to be used every three or four search iterations.

4.5 Other pruning techniques

There are several pruning techniques particular to Model Elimination that we incorporate in METEOR and which are used in other ME based provers ([28], [32], [8].) The most useful of these prunes (with failure) a branch of the search tree if the goal (literal eligible for extension and reduction) is identical to an ancestor literal. The other techniques prune (with success) a branch of the search tree if either reduction or extension with a unit chain does not specialize any variables in the goal. (These pruning rules are derived from Definition 2.1, the definition of an admissible chain.) The first of these techniques has a dramatic effect on the time needed to find a proof for many of the problems. In METEOR, the literals of a chain with the same sign and predicate are linked together so that this pruning method is relatively inexpensive to implement. The other techniques are also implemented efficiently due to the use of the trail stack. If a reduction or unit extension does not specialize any variables, no bindings will be trailed and a simple check of whether the top of the trail stack has changed is sufficient to determine whether pruning is permissible.

These pruning measures are active by default in METEOR although they can be deactivated at runtime. Note that in a parallel setting it is possible for the pruning techniques based on variable instantiation to be voided if a processor steals work prior to pruning.

Another pruning technique involves limiting the number of times a clause can participate in a deduction. This technique proved quite useful in the earliest implementation of Model Elimination [12] and appears to have been investigated by the PARTHEO group [28]. In METEOR, it is possible to limit the number of times a clause is used in a deduction by either annotating each clause in the input file (and specifying the appropriate runtime parameters) or by uniformly limiting clause use at runtime. The limit is incremented during each iteration of the search as are the other depth measures.

We also have implemented a mechanism that can control the complexity of the terms generated during a deduction. This is often useful in problems involving complex terms (e.g., the Bledsoe problems below) even though our mechanism is inefficient and reduces the inference rate by a significant factor. Our implementation is much cruder, for example, than that employed by Otter [23]. We do not employ this pruning method nor the clause limiting method in the results reported here.

5 Results

In this section we provide the results of running METEOR on several problems from the literature using several different depth measures. We also compare these figures to those given for several other comparable theorem provers.

In these tables *sMETEOR* refers to our sequential implementation of METEOR. Statistics for this implementation are calculated from an average of at least five runs on a Sun SPARC Station 1+. Statistics for the parallel implementation of METEOR are also averaged over five runs on a Butterfly GP1000 using up to 30 nodes. (Each node consists of a Motorola 68020 processor, four megabytes of memory, and some custom hardware. There is no user level intranode multiprogramming.) We note that many of the problems can yield substantially different run times over several runs using the same number of processors due to the non-deterministic nature of parallel computation. Our distributed prover, *dMETEOR*, runs on a network of Sun SPARC and Sun 3 workstations. Since each "processor" in this network has a potentially different workload (unlike the Butterfly), and since our prover runs in a low priority background mode, the number of CPU seconds per processor varies greatly (across processors) during a distributed run.

Unless otherwise indicated, the problems we have used in testing our prover come from [32] (which pulled together problems from many other works, principally from Wilson and Minker [35]) and have been used as reported benchmarks in several other provers [8] [27].

It is difficult to compare the results reported for different automated reasoning systems because of differences in hardware. Ideally we would run the provers referenced in the tables below on the same machine and compare the runtimes, but we do not have access to all the provers nor are we expert at the use of many of the different provers. Nevertheless, we feel it is useful to report the comparisons we have made.

5.1 Bledsoe challenge problems

We have begun an investigation of the proofs of the problems Woody Bledsoe recently re-introduced [5]. The proofs of successive problems in this series are progressively deeper and harder. We have been able to prove the first two problems relatively easily with the appropriate depth measures, but have had trouble with the third problem in the series. The proof tree for the first problem is given in Figure 6.

The boxed regions of Figure 6 enclose the same proof tree. This proof tree can be developed only once and retained as a lemma. (The numbers of the nodes refer to clause numbers used in [5].) Similar multiple occurrences of subtrees, hence opportunity for lemma use, occur in proofs of the second and third problem. The proof search for the first and second Bledsoe challenge problems did not utilize lemmas. The third challenge problems has proven inaccessible to straight runs under any of the discussed depth measures, but has been solved when a lemma was used as an axiom. Likewise, we have been able to prove the lemma when stated as a goal (see Table 14.) This is in no manner a demonstration that we can "solve" the third problem because lemma generation under several filters we have tried still returns too many lemmas to automate the passage from lemma generation to lemma use. (For the first two challenge problems, lemma generation is not so bountiful, but we have not yet explored coupling lemma generation and use systematically in any setting.) The third challenge problem illustrates the potential for added power if lemma selection can be done successfully for some deeper problems. However, this third problem also demonstrates the limitation of purely high inference rates as this problem is combinatorially beyond the reach of linear input procedures without some augmentation similar to the use of lemmas.

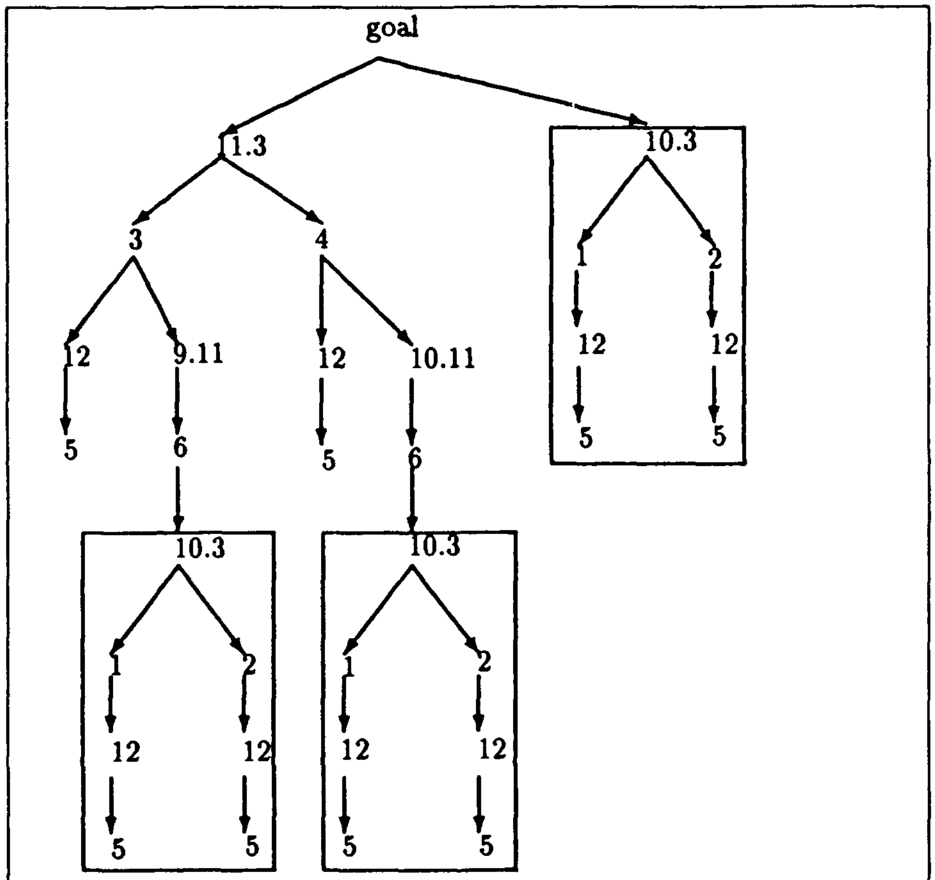

Figure 6: Proof tree for first Bledsoe challenge problem

In Figures 11 and 12 we provide figures for several problems run on the Butterfly GP1000. The "superlinear" speedup (speedup of more than *n* on *n* processors) arises from the search at the last iteration of iterative deepening. If one of *n* processors chooses an alternative near the root of the search tree that leads directly to a proof, the time of this last iteration will be significantly less than the time of the last iteration using one processor. If completely searching all levels of the search tree prior to the last iteration is done efficiently, then a superlinear speedup results.

problem	Time (seconds) for several provers prover				
	PARTHEO†	PTTP	PARTHENON‡	sMETEOR	METEOR*
apabhp	na	873	2886/292	146.4	1,986/22
ls36	1666/352	548	2273/151	137.9	1,820/41
wos1	4.6/1.1	57.6	151/12	26.9	318/10.3
wos4	404/1.3	4,152	14,000/501	0.78	10.3/1.1
wos10	263/136	29.4	159/20	17.2	227/6.6
wos21	na	966	3315/373	503.8	6,373/274

†ME-depth, no iterative deepening, 1/16 processors
‡1/15 processors
* 1/30 processors

Figure 7: Comparison of different provers

Comparison with non-linear provers (runtime in seconds)			
problem	Otter†	Lee [27]‡	METEOR§
apabhp	24	748	3.6
ls36	< 1	56.5	138
wos10	$< .5$	35.9	13.5
wos15	3	3,549	14,546
wos21	16	2,005	504
wos4	< 1	20.4	< 1
wos22	¶	73	381
pigeon 5	¶	7.28	25.67
ex 5	¶	97.9	15.62

†on Sun SPARC
‡on Sun 3
§optimal non-weighted depth measure
¶memory limit exceeded

Figure 8: Comparison with non-linear provers

problem	sMETEOR Timings (seconds) Depth Measure Used			
	cumulative inferences	ME-depth (# A-literals)	Heuristic (A-lits = inf/2)	Weighted
apabhp	146.4	3.6	16.6	1.1
ls36	137.9	31,023	109.4	11.1
wos1	26.9	3.1	3.9	0.7
wos4	0.78†	0.18	0.87†	0.87
wos10	17.2	206	13.48	9.2
wos21	503.8	> 190,600	344	249
bledsoe1		2,333		2.1
bledsoe2		7,984		194.09
steamroller	398†	3.41	400†	1.63

†non-horn, rollback reduction used

Figure 9: sMETEOR Timings (seconds) with different depth measures

problem	Successful Inferences/second prover				
	PTTP	PARTHENON †	sMETEOR	wMETEOR	METEOR‡
apabhp	1,957	605/7,478	3,634	5,118	549/16,813
ls36	2,555	644/8,197	5,028	4,235	465/14,000
wos1	2,413	914/9,465	5,170	4,409	449/12,516
wos4	2,182	705/9,532	4,735	4,648	404/2,996
wos10	2,675	619/8,419	4,575	3,603	392/12,318
wos21	2,481	778/10,023	4,760	4,305	407/12,050
bledsoe1			1,224	1,682	440
bledsoe2			1,550	2,691	326
steamroller			10,943	12,403	960

†1/15 processors
‡1/30 processors

Figure 10: Successful inferences per second

problem	Time in seconds (speedup) Cumulative Inference Depth Measure Number of processors				
	1	2	10	20	30
apabhp	1,986	341 (5.82)	61 (32.55)	32 (62.06)	22 (90.27)
ls36	1,820	888 (2.04)	99 (18.38)	55 (33.09)	41 (44.39)
wos1	318	153 (2.07)	29 (10.96)	15 (21.2)	10.5 (30.28)
wos4	10.3	6 (1.71)	2.28 (4.51)	1.33 (7.74)	1.1 (9.36)
wos10	227	112 (2.02)	21 (10.8)	13 (17.46)	6.6 (34.39)
wos21	6,373	3290 (1.93)	848 (7.51)	335 (19.02)	274 (23.25)

Figure 11: Butterfly Execution Results (cumulative depth measure)

problem	Time in seconds (speedup) Heuristic Depth Measure Number of processors				
	1	2	10	20	30
apabhp	234.8	40.17 (5.84)	8.1 (28.98)	5.75 (40.83)	4.32 (54.35)
ls36	1,483	724.7 (2.04)	78.7 (18.84)	45.6 (32.52)	35.2 (42.13)
wos1	53.3	21.85 (2.43)	4.36 (12.22)	3.6 (14.8)	3.0 (17.76)
wos10	188	92.6 (2.03)	20.8 (9.03)	6.8 (27.64)	6.06 (31.02)
wos21	4,678	2,396 (1.95)	680.5 (6.87)	248 (18.86)	204 (22.93)

Figure 12: Butterfly Execution Runtime (heuristic depth measure)

problem	Some harder problems time (secs)	depth measure	prover
wos22	381.14	ME-depth	sMeteor
	11,703	heuristic	sMeteor
	17,797	inference	sMeteor
	26,766	inference	Butterfly†
wos15	14,546	inference	sMeteor
	10,604	heuristic	sMeteor
	5,629	heuristic	Butterfly †
	non-horn problems		
ex5	16,889	heuristic	sMeteor
	1,662	ME-depth	sMeteor
	15.62	(no reductions)	sMeteor
wos4	1,083	inference	sMeteor
	0.78	rollback	sMeteor
steamroller	3.41	ME-depth	sMeteor
	398	rollback	sMeteor
	38.85/2.97	ME-depth	Butterfly ‡
pigeon 3	0.04	ME-depth	sMeteor
	223.9	inference	sMeteor
pigeon 4	0.51	ME-depth	sMeteor
	1.21	rollback	sMeteor
pigeon 5	25.67	ME-depth	sMeteor
nonobvious	1.51	ME-depth	sMeteor
(see [26])	7.41	rollback	sMeteor

†20 processors
‡1/20 processors

Figure 13: Some harder problems

Bledsoe Third Challenge Problem (using lemma as axiom)		
time (secs)	prover	depth measure
31,649	sMETEOR	ME-depth
≈ 70	dMETEOR (10 procs)	ME-depth
165	METEOR (20 procs)	ME-depth

Figure 14: Bledsoe third challenge problem

6 Clause Sets

6.1 wos1

This is an example from group theory. Given left identity and inverses, there is a right identity. In the clauses below, **e** represents the left identity, **g** is the left inverse function, **h** is the right inverse function, and **f** represents the result of operating on two group elements. The predicates used are **p** for the result of operating on two objects ("product") and **r** for equality of two objects. Taken from [35].

```
1.  p(e,X,X)
2.  p(g(X),X,e)
3.  p(X,Y,f(X,Y))

4.  r(X,X)

5.  -p(X,Y,U), -p(Y,Z,V), -p(U,Z,W), p(X,V,W)
6.  -p(X,Y,U), -p(Y,Z,V), -p(X,V,W), p(U,Z,W)

7.  -r(X,Y), r(Y,X)
8.  -r(X,Y), -r(Y,Z), r(X,Z)

9.  -p(X,Y,U), -p(X,Y,V), r(U,V)
10. -r(U,V),   -p(X,Y,U), p(X,Y,V)
11. -r(U,V),   -p(X,U,Y), p(X,V,Y)
12. -r(U,V),   -p(U,X,Y), p(V,X,Y)

13. -r(U,V), r(f(X,U),f(X,V))
14. -r(U,V), r(f(U,Y),f(V,Y))
15. -r(U,V), r(g(U),g(V))
16. -r(U,V), r(h(U),h(V))

goal
17. -p(h(X),X,h(X))
```

6.2 wos10

Another group theory problem. The constants, functors, and predicates are as in wos1. This is a proof of: if every group element is idempotent (e.g., X*X = e), then the group is commutative.

```
1.  p(e,X,X)
2.  p(g(X),X,e)
3.  p(X,Y,f(X,Y))
4.  p(X,e,X)
5.  p(X,g(X),e)
6.  p(X,X,e)
7.  p(a,b,c)

8.  r(X,X)

9.  -p(X,Y,U), -p(Y,Z,V), -p(U,Z,W), p(X,V,W)
10. -p(X,Y,U), -p(Y,Z,V), -p(X,V,W), p(U,Z,W)

11. -r(X,Y), r(Y,X)
12. -r(X,Y), -r(Y,Z), r(X,Z)

13. -p(X,Y,U), -p(X,Y,V), r(U,V)
14. -r(U,V),   -p(X,Y,U), p(X,Y,V)
15. -r(U,V),   -p(X,U,Y), p(X,V,Y)
16. -r(U,V),   -p(U,X,Y), p(V,X,Y)

17. -r(U,V), r(f(X,U),f(X,V))
18. -r(U,V), r(f(U,Y),f(V,Y))
19. -r(U,V), r(g(U),g(V))

goal
20. -p(b,a,c)
```

References

[1] Khayri A. M. Ali and Roland Karlsson. The Muse Or-Parallel Prolog Model and its Performance. In *North American Conference on Logic Programming*, pages 757-776, 1990.

[2] Owen Astrachan. METEOR: Model Elimination Theorem-proving for Efficient OR-parallelism. Master's thesis, Duke University, 1989.

[3] R. Bayer and M. Schkolnick. Concurrency of operations in b-trees. *Acta Informatica*, 9:1-21, 1977.

[4] W. Bibel. Short Proofs of the Pigeonhole Formulas Based on the Connection Method. *Journal of Automated Reasoning*, 6:287–297, 1990.

[5] W. W. Bledsoe. Challenge Problems in Elementary Calculus. *Journal of Automated Reasoning*, 6(3):341–359, 1990.

[6] W.W. Bledsoe and L. Hines. Variable Elimination and Chaining in a Resolution-Based Prover for Inequalities. In *Fifth Conference on Automated Deduction*, pages 281–292. Springer-Verlag, 1980.

[7] Soumitra Bose, Edmund Clarke, David E. Long, and Spiro Michaylov. Parthenon: A parallel theorem prover for non-Horn clauses. In *Symposium on Logic in Computer Science*, 1989.

[8] Soumitra Bose, Edmund Clarke, David E. Long, and Spiro Michaylov. Parthenon: A parallel theorem prover for non-Horn clauses. *Journal of Automated Reasoning*, 1990. (to appear).

[9] R.S. Boyer and J. Moore. The sharing of structure in theorem proving programs. In B. Meltzer and D. Michie, editors, *Machine Intelligence 2*, pages 101–116. Edinburgh University Press, 1972.

[10] C. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[11] S.A. Cook and R.A. Reckhow. On the lengths of proofs in the propositional calculus. *Journal of Symbolic Logic*, 44(1):15–22, 1979.

[12] S. Fleisig, D. Loveland, A. Smiley, and D. Yarmash. An implementation of the model eliminiation proof procedure. *JACM*, 21:124–139, January 1974.

[13] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

[14] Richard E. Korf. Depth-first iterative deepening: An optimal admissible tree search. *Articial Intelligence*, 27:97–109, 1985.

[15] David Kotz and Carla Ellis. Evaluation of concurrent pools. In *Ninth Internation Conference on Distributed Computing Systems*, pages 378–385, 1989.

[16] R.A. Kowalski and D. Kuehner. Linear Resolution with Selection Function. *Artificial Intelligence*, 2:227–260, 1971.

[17] Vipin Kumar and V. Nageshwara Rao. Scalable Parallel Formulations of Depth-first Search. In Vipin Kumar, P.S. Gopalakrishnan, and Laveen N. Kanal, editors, *Parallel Algorithms for Machine Intelligence and Vision*, pages 1–41. Springer-Verlag, 1990.

[18] Donald W. Loveland. Mechanical theorem proving by model elimination. *JACM*, 15(2):236–251, April 1968.

[19] Donald W. Loveland. A simplified format for the model elimination procedure. *JACM*, 16(3):349–363, July 1969.

[20] Donald W. Loveland. *Automated Theorem Proving: A Logical Basis*. North-Holland, 1978.

[21] Ewing Lusk, David H. D. Warren, and Seif Haridi et al. The Aurora Or-parallel Prolog System. *New Generation Computing*, 7:243–271, 1990.

[22] David Maier and David S. Warren. *Computing with Logic*. Benjamin/Cummings, 1988.

[23] William W. McCune. *OTTER 2.0 Users Guide*. Argonne National Laboratory, March 1990.

[24] Nils Nilsson. *Principles of Artificial Intelligence*. Tioga Press, 1980.

[25] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

[26] F.J. Pelletier and P. Rudnicki. Non-obviousness. *AAR Newsletter*, (6):4–5, 1986.

[27] David Plaisted and Shie-Jue Lee. Inference by clause linking. Technical Report TR90-022, University of North Carolina, Department of Computer Science. Chapel Hill, NC, 1990.

[28] J. Schumann and R. Letz. PARTHEO: A High Performance Parallel Theorem Prover. In *Tenth International Conference on Automated Deduction*, pages 40–56, 1990.

[29] Mark E. Stickel. A prolog technology theorem prover. *New Generation Computing*, 2(4):371–383, 1984.

[30] Mark E. Stickel. A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Ccompiler. In *Eight International Conference on Automated Deduction*, pages 573–587. Springer-Verlag, 1986.

[31] Mark E. Stickel. Schubert's Steamroller Problem: Formulations and Solutions. *Journal of Automated Reasoning*, 2:89–100, 1986.

[32] Mark E. Stickel. A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Ccompiler. *Journal of Automated Reasoning*, 4:343–380, 1988.

[33] David H.D. Warren. An abstract prolog instruction set. Technical Report 309, SRI International, Menlo Park, California, October 1983.

[34] David H.D. Warren. The SRI model for OR-Parallel execution of Prolog — abstract design and implementation issues. In *IEEE Symposium on Logic Programming*, pages 92–102, 1987.

[35] Gerald A. Wilson and Jack Minker. Resolution, Refinements, and Search Strategies: A Comparative Study. *IEEE Transactions on Computers*, C-25(8):782–801, August 1976.